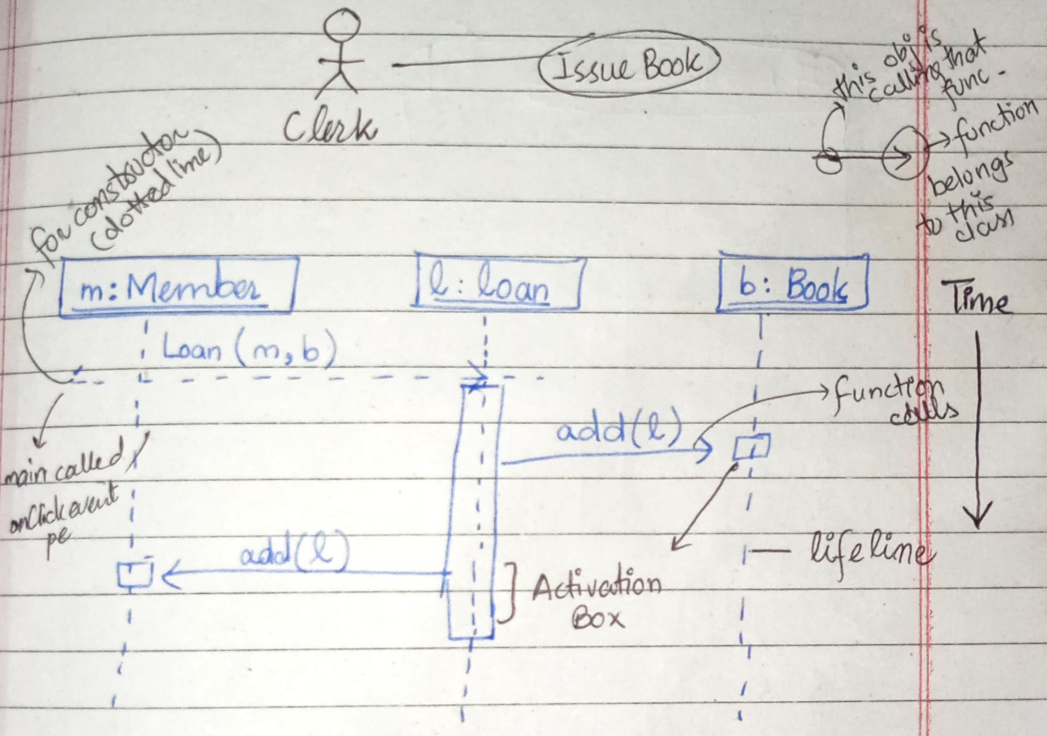
Lecture # 11

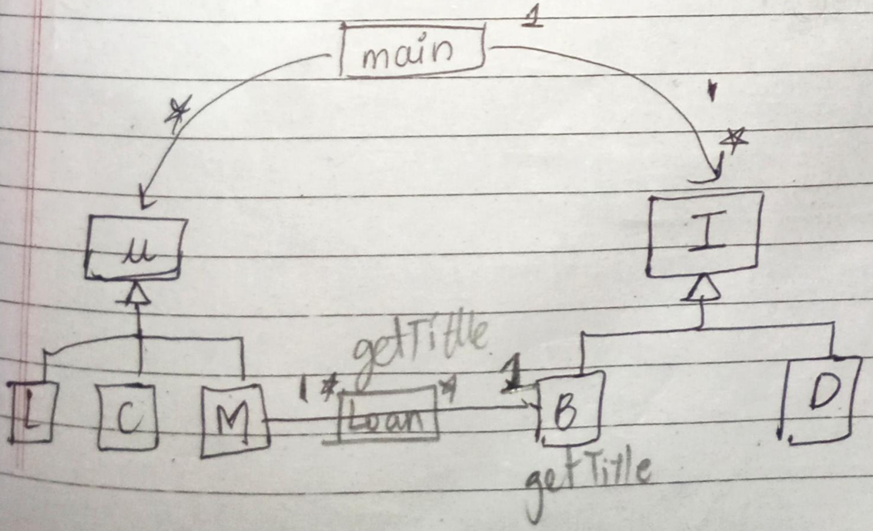
Date: Sep 29, 25 Day: Monday

Sequence Diagram:

A SD shows the object interactions (function calls) required to execute a use case (operation)

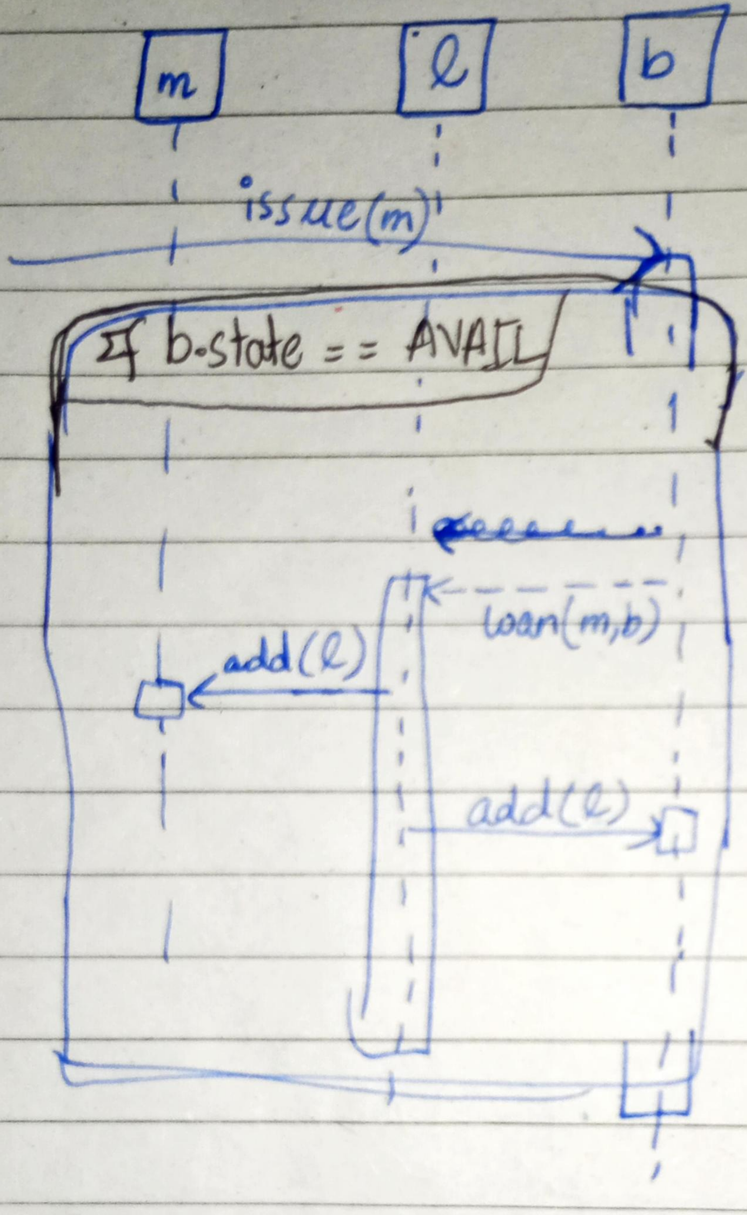


Clerk
Issue Book
this obj is calling that func.
function belongs to this class
for constructor (dotted line)
m: Member
l: loan
b: Book
Time
Loan (m, b)
main called / onClick event pe
add(l)
function calls
add(l)
lifeline
Activation Box



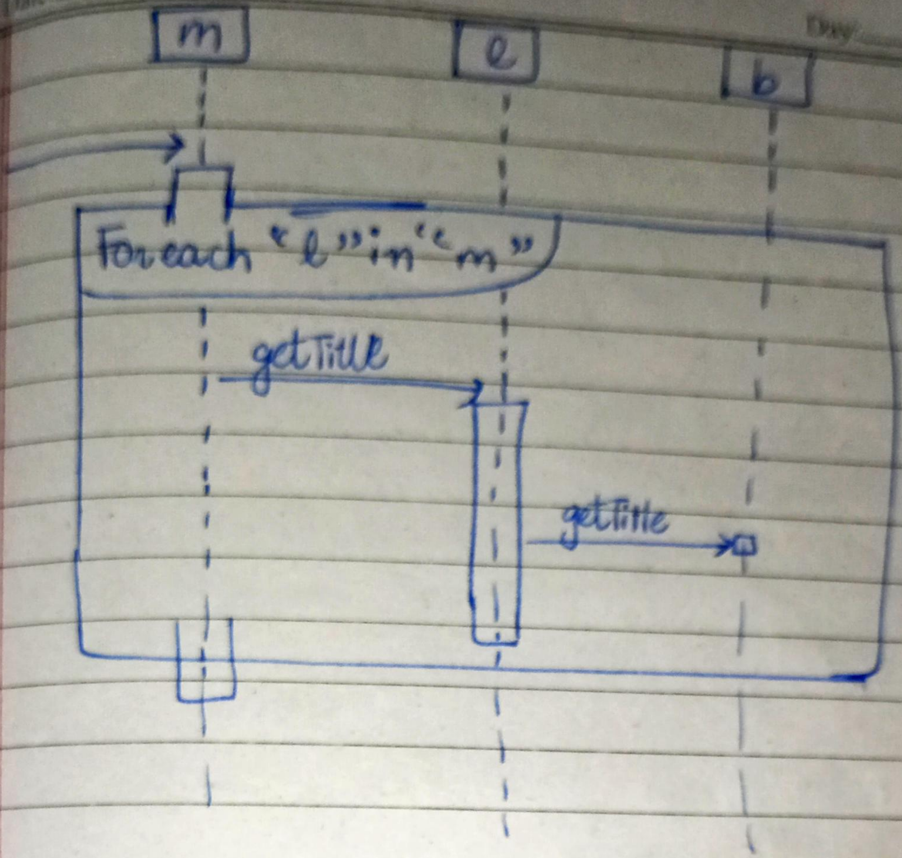
If some books are issuable and some are not:

b1.issue(m1);
b2.issue(m1);



Task:

Give a SD to print titles of all the books borrowed by a particular member.



Wednesday! Lecture # 12 Oct 1st, 2025

Cohesion and Coupling:

Cohesion is low when we pack multiple entities or tasks into a single abstraction.
(function or class)

Two Design Principles:

1- High cohesion → (functions/classes focuses on one thing).
2- Low coupling.

For example:

→ searchAndSort (___)
→ class studAndTA

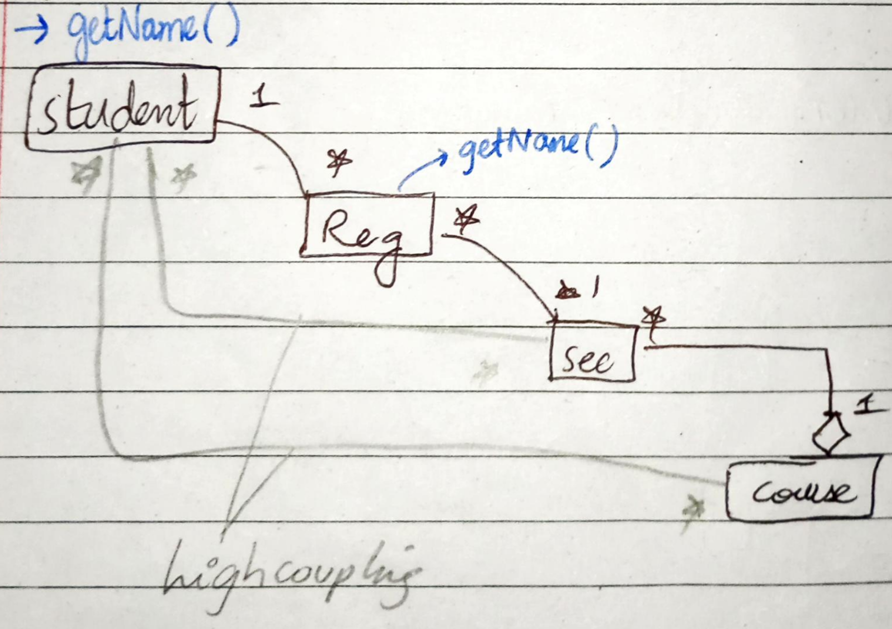
```
→ int average(---){
        - - - -
        - - - -

      cout << avg;

      }
```

✯ Low coupling:
↳ Coupling is the degree of interaction b/w different moules(functions, classes, components, etc).

→ Associations → Function calls.
→ Inheritance



* Don't "talk" to your neighbor's neighbor
(Demeter's Law)

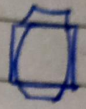
Date:

Low coupling

Reg1 ←
↘ Reg2

Day:

```
class sec {
        Reg * a[N];
  void printStudents( ) {
  for (i = 0   to  N) {
     stud_id = a[i]. getStudId( );
                ①  . getName( );
     ~~stud~~
             }
```



referencing one class only.

```
class Reg {
        student * st ;
        Sec * sec
      char * getName( ) {
            return st→getName( );
           }   → student * getStud( ) {
                     return this→st;
                  }
```

```
void printStudent ( ) {

    for (int i = 0 ; i <N ; i++){
    student * s = a[i] → get Stud( );
                   ①

           sout << s→ get Name( );
                       ②

           }
```

→ high coupling
→ referencing two classes.